EMUL12-PC/BDM

User Guide

ICE TECHNOLOGY nohau®

EMUL12-PC/BDM

User Guide

ICE Technology
Tel: 800.686.6428
Fax: 650.375.0409
URL: http://www.icetech.com
E-Mail: support@icetech.com

Edition 3

Development	***Documentation***
Doron Fael Jörgen Andersson	Doron Fael

We would appreciate any feedback about the product (including the manual) ranging from simple software defects to suggestions on how to improve the examples.

Thank You.

Warranty Information

The EMUL12-PC/BDM pod, EMUL-PC/LC-ISA board, and emulator cable are sold with a one-year warranty starting from the date of purchase. Defective components under warranty will either be repaired or replaced at ICE Technology's discretion.

The EMUL12-PC/BDM Windows Emulation software is sold with no warranty, but upgrades will be distributed to all customers up to one year from the date of purchase. ICE Technology makes no other warranties, express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. In no event will ICE Technology be liable for consequential damages. Third-party software sold by ICE Technology carries the manufacturer's warranty

Table of Contents

Overview

Introduction to EMUL12-PC/BDM

The EMUL12-PC/BDM is a personal computer-based Background Debug Mode emulator for Motorola's 68HC12 16-bit family of microcontrollers. EMUL12-PC/BDM consists of a small PC plug-in board, a five foot long 25 pin cable, a BDM "pod", a 6 conductor ribbon cable with a BERG connector in the end, and two wires with receptacles. Optional target boards are available for most HC12 controllers that support Background Debug Mode. The EMUL12-PC/BDM supports all present and future Motorola microcontrollers that incorporate CPU12, and that utilize the single wire Background Debug Mode concept, which includes the 68HC12 family of microcontrollers.

Note: EMUL12-PC/BDM is not a simulator and has no 68HC12 family controller of it's own. It requires a working target board of some kind with a Motorola-standard 6 pin BERG connector. To load program to the target it requires enough RAM on your target (you may use the on chip RAM) to download your program. You may also program the internal EEPROM and FLASH memories (if they exist on your target) using the EMUL12-PC/BDM.

The EMUL12-PC/BDM software is a Microsoft *Windows* 3.x, 95, NT application. It follows the MS *Windows* Multiple Document Interface Standard, resulting in the same look and feel as applications produced by Microsoft and others for MS *Windows*.

The EMUL12-PC/BDM user interface is consistent with most other MS *Windows* applications and includes dynamically changing menus, moveable and scrollable “child” windows, function key shortcuts for menu items, and context sensitive help. Anyone familiar with MS *Windows* applications will be able to use EMUL12-PC/BDM with little or no other assistance.

The EMUL12-PC/BDM hardware is modular. The software user interface implements an effective high level debugger. It has support for local variables, C typedefs, and C structures.

How to use this manual

This manual was written with different kinds of users in mind. All users should have MS *Windows* installed and have learned the skills taught in the Basic Skills chapter of the *Microsoft Windows User's Guide*. It also assumes a basic familiarity with the chip you are using. Many of the EMUL12-PC/BDM features are designed around the features of the supported chips. Being familiar with the chip makes it easier to understand how to use the emulator productively.

If you are new to emulators of any kind,

read the manual completely, including the reference chapters. You may skip the sections that describe target processors you do not have.

If you have used emulators with other microprocessors

and understand the difference between a hardware breakpoint and a software breakpoint, but are not familiar with the chip being emulated, you are strongly encouraged to review the features of the chip you have, then run the emulator.

If you are quite familiar with emulators, MS Windows, and the chip,

read the “Setting Up the Hardware” and “Software User Interface” chapters, and then begin using EMUL12-PC/BDM, referring to on-line help when needed. After a few days of use, skimming the Reference chapters may highlight useful features.

Manual Conventions

Type the words in double quotes exactly as shown (without the quotations) except for the <Enter>, <Ctrl>, <Tab>, and <Alt> keys. Use the <Alt> and <Ctrl> keys like shift keys. Hold them down while you press the keys that follows them in the text. For example, if the text below instructs you to type <Alt>F, press down and hold down the <Alt> key then press the F key.

Window names and labels that appear on the screen are printed using a font that closely matches the screen font.

Notes and *hints* are printed in *italics* and **warnings** have a box around them to set them apart from the rest of the manual text. Pay careful attention to all **warnings**.

Quick Installation Instructions

System Requirements

EMUL12-PC/BDM requires a personal computer with at least one free ISA-bus slot. The P.C. must also have at least 2 megabytes of RAM, a CPU that is a 386 (or higher) PC compatible, a hard disk with at least 3 megabytes of unused space and Microsoft *Windows* 3.x / '95 / NT installed. A mouse is not required, but is strongly recommended.

Quick Setup Instructions

The hardware and software are designed to be easily installed and quickly running on most personal computer systems. Users can normally begin using their emulator after following these few steps. However, if you are new to personal computers, if you are unsure about what to do after reading these quick installations instructions, or if your emulator does not work after you follow these instructions, follow the steps for installing and configuring each board and software as outlined in their respective chapters.

Installing the Emulator

Installing the emulator board is much like installing most other AT-style boards.

1. Turn off the power.
2. Remove the P.C. cover.
3. Remove the slot cover, if present, for an available 8 bit slot.
4. Insert the EMUL-PC/LC-ISA board into the slot, use a screw to secure the board
5. You can now close the cover.

Installing the Cable

The DB-25 is a keyed connector. That means it cannot be plugged in incorrectly. Plug the 25 pin DB connector into the 25 pin DB connector on the PC plug-in board. Likewise, plug the other end of the cable into the BDM pod.

Connecting the BDM pod to your target

The 6 pin BERG connector at the end of the 5” ribbon cable coming out of the BDM pod is not keyed. Wire 1 on the ribbon connector is the red wire. The socket for pin 1 on the plug is on the same side of the connector as wire 1 (the red wire) on the ribbon cable (refer to page 38 for connector pin-out). The 2 extra wires (the yellow wire and the red wire) are to be connected to the MODA (yellow wire) and MODB (red wire) pins of the target HC12 chip. Make sure the target board does not drive the RESET, BKGD,

MODA and MODB pins of the target. These pins are connected to the BDM pod (for further details refer to page 38).

Installing the Software

Initial Software Configuration - Emulator Hardware Configuration

The first time you activate the EMUL12-PC/BDM software, the following **Hardware Setup** window will appear:

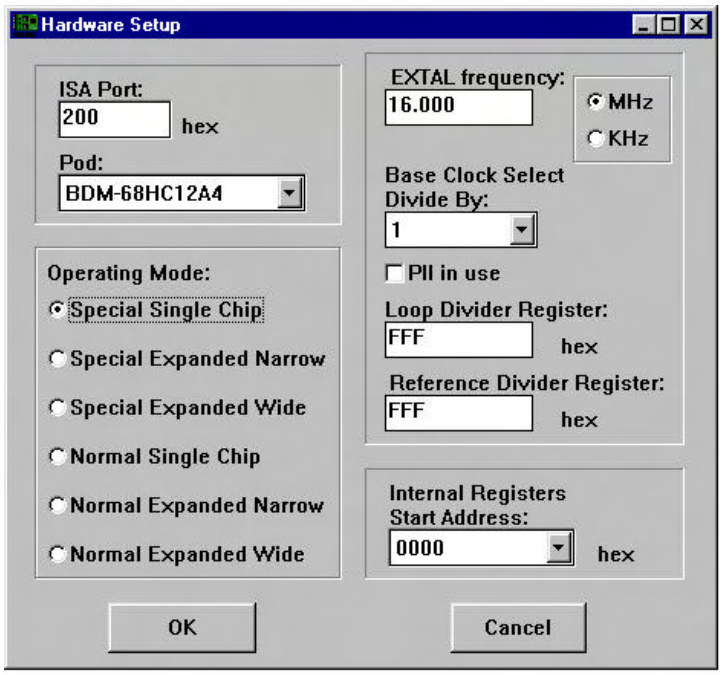


Figure 1: Hardware Setup Window

At this point, you are asked to initialize the following parameters:

1. ISA port address of the PC plug-in board (factory jumpers configured to 200H).

Note: If you need to map the emulator to another port address, refer to page 35 for instructions how to set the jumpers on the PC plug-in board.

2. Select the type of your target processor in the Pod field.

3. Select the target Operating Mode you would like to work in, using the Operating Mode radio buttons.

4. Type in the external target crystal or oscillator frequency that is received on the EXTAL pin of your target (this sets the requested communication rate of the BDM channel between the BDM pod and your target). The typed number is in units of KHz or MHz depending on the radio buttons on the right of the Extal Frequency field. If the frequency is in the MHz range, check the MHz radio button; if the frequency is in the KHz range, select the KHz radio button. In

Figure 1 above, for example, the target external crystal frequency is set to 16.0 MHz.

Note: The allowed input frequency range for the EMUL12-PC/BDM is 32KHz - 32MHz, and the allowed ECLK frequency range is 16KHz - 16MHz. Trying to work in a frequency outside of this range will fail.

5. If you are using an HC12 controller where the CLKCTL register exists, and Base Clock Select bits (BCSC - BCSA) exist, select the requested clock divide to be written to these bits in the 'Base Clock Select Divide By' field.

Note: Your application may not change the BCSC - BCSA bits of the CLKCTL register. You may write these bits with the same value as requested here, but if you write a different value to these bits, a loss of communication between your target and the BDM pod may occur, and the only way to overcome this may be to reset the target.

6. If you are using an HC12 controller that includes a PLL on chip, check the 'Pll in use' box if you would like to use the on-chip pll. The EMUL12 software will activate the target PLL and connect it to be your target clock source (if you check the 'Pll in use' box).

Note: Your application may not activate or deactivate the target pll unless the pll output frequency is identical to the operating frequency when the pll output is not selected.

7. If you are using an HC12 controller that includes a PLL on chip, and you would like to use the target on chip pll, type the requested values for the Loop Divider register (LDV) in the 'Loop Divider Register' field, and the requested values for the Reference Divider register (RDV) in the 'Reference Divider Register' field. The EMUL12 software will write these two registers with the specified values for you.

Note: Your application may not change the content of the LDV & RDV registers if the on chip PLL is in use. It may, however write these registers with the same values you have typed in here. Writing a different value to these registers may cause a loss of communication between your target and the BDM pod, and the only way to overcome this may be to reset the target.

8. Select the requested starting address of your target internal registers in the 'Internal Registers Start Address' field. The EMUL12 software will write the appropriate value according to this field to the INITRG register of your target.

Note: Your application software may not change the INITRG register value. It may, however, write the same value as initialized in this field. Writing another value to the INITRG register will disable the EMUL12 software from supporting different features which require writing and reading of the HC12 registers.

9. Click on the 'OK' button to approve the new setting you have just filled and activate the EMUL12 user interface software. This action also saves the new options in the emulhc12.ini file, so next time you enter this program or reset the target chip, the EMUL12 software will automatically read the settings from the

emulhc12.ini file and implement them. The EMUL12 user interface program will use the new setting and implement them in your target chip.

10. Clicking on the 'Cancel' button will disregard the new setting you have just filled and activate the EMUL12 user interface software with the old setting of the emulhc12.ini file.

11. You can always re-activate the **Hardware Setup** window by selecting the **Emulator Hardware** item in the **Config Menu**.

Quick Start Instructions

This section describes how to quickly start using EMUL12-PC/BDM to debug an existing program or target board once the EMUL12-PC/BDM hardware is installed, the user interface software is running, and the **Hardware Setup** window is set correctly.

To load and execute a program:

1. Select **Load code ..** from the **File** menu and identify the file using the dialog box

Note: If you do not have code to load, do the following to enter your own small user program:

*click in the **Program** Window*
type <Ctrl>A
type in address 800
type <Ctrl>N to force the program counter to address 800
hit <Enter>
type: NOP <Enter>
NOP <Enter>
JMP $800
then continue with item 2 below.

2. Click on the Go button in the tool bar

3. Click on the Break button in the tool bar.

5. Repeat steps 2 and 3 several times and observe the software break in another line each time.

To set a software breakpoint either:

1. Make sure the program is breaked first.

2. Click twice on the desired instruction in any **Program** window, or

3. Click once on the line number for the desired instruction in any **Source** window.

To make a software breakpoint inactive either:

1. Click on the desired breakpoint in any **Program** or **Source** window, then press F2, or...

2. Select **Setup ..** from the **Breakpoints** menu, click on the breakpoint, click on the Toggle button, or double click on the breakpoint, or…
3. Highlight (click once on) the breakpoint and select **Toggle Breakpoint** from the **Program** or **Source** menu or...
4. Highlight (click once on) the breakpoint and select **Toggle** from the **Breakpoints** menu, or…
5. Click on the line where you would like to inactivate a breakpoint, and then click again on the same line.

To delete a software breakpoint either:

1. Select **Setup ..** from the **Breakpoints** menu, click on the breakpoint and click on the **Delete** button or...
2. Select **Delete All** from the **Breakpoints** menu

To use the in-line assembler to change the program loaded:

1. Scroll a **Program** window to show the address to be changed
2. Highlight the instruction to be changed with the cursor or arrow keys
3. Type the desired mnemonic (this will open an **Enter data** dialog box) and hit <Return>

To change a RAM value:

1. Scroll a **Data** window to show the address to be changed
2. Highlight the address to be changed with the cursor or arrow keys
3. Type the desired value (this will open an **Enter data** dialog box) and hit <Return>

Chapter 1: Software User Interface

Detailed Software Installation Instructions

Before installing the software, it is important to have a basic understanding of how to operate *MS Windows*. For help mastering *MS Windows*, please refer to the *Microsoft Windows User's Guide.*

The EMUL12™-PC/BDM floppy disk includes an *MS Windows* compatible SETUP.EXE program. To install this software, run SETUP.EXE by typing "WIN A:SETUP" at the DOS prompt before entering *Windows* or, from within *MS Windows*, by selecting the **RUN** item in the **Program Manager File** menu and selecting "A:SETUP" as the file to run. A dialog box will ask for a directory for the EMUL12™-PC/BDM software. You will be asked if you are using Windows NT on your PC; check if you are. Either accept the suggested directory or type a different one. SETUP will copy files from the floppy to the hard disk directory specified and change the various *MS Windows* ".ini" files as needed. When installed, there will be an **EMUL12** program group with an icon labeled **EMUL12**. Double-clicking on the emulator icon will start the EMUL12™-PC/BDM application. If you wish to move the icon to another group, you may do so by using the **Move...** menu item in the **Program Manager's File** menu or by dragging the icon to the new group.

Initial Software Configuration

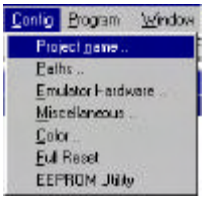
The type of target processor in the hardware setup must agree with the type of target you are using. If not, you may see an error message. To ensure that you do not get this error, use the **Hardware Setup** window that is accessed when you first install your emulator, and possibly again when you change your target type. (You can also activate this window any time you want to check the values in the initialization file.) The **Hardware Setup** window appears automatically when you run the EMUL12-PC/BDM application for the first time, and you can always reactivate it by selecting the **Emulator Hardware** item from the **Config** menu.

Refer to the “Initial Software Configuration - Emulator Hardware Configuration” section on page 4 for a detailed description on how to initialize the software using the **Hardware Setup** window.

Configuring the Software

If the Quick Installation instructions do not work, you will most likely need to adjust either the hardware jumpers, the software configuration, or possibly both. It is also possible that if your target processor is not working for some reason, you will receive an error message. Please refer to the appropriate chapters for setting the jumpers on the PC plug-in board (the EMUL-PC/LC-ISA board). The next few pages describe all of the items in the **Config** menu. Use these menu items to examine the software configuration in detail and to change it if needed.

Projects



A project is a collection of software configuration settings that are all associated with a specific person, target, or software development project. This menu item opens a dialog box that allows you to set up named configurations or projects. This is first in the menu and described first because all of the other Config menu item settings will be stored as settings for the current project in a file with a ".PRO" suffix. There is an ".INI" file and those settings are used if there is no current project. But if the ".INI" file contains the name of the current project, all software settings are taken from ".PRO" file for that project.

Projects behave differently than, for instance, a word processing document. All software configuration settings are written to disk every time you change projects or whenever you exit the emulator software. There is no "exit without saving changes" option. Once you make a change to the configuration, it is immediately effective and will, unless you manually undo the change, be saved to the disk in the project file.

Creating a Project

Users who change the software settings and THEN change the name of the project may believe the old project will remain unchanged. In fact, the moment a new project is created, the current settings will be saved to the old project, not the new project. The new project will be saved when exiting the debugger or when changing projects (again).

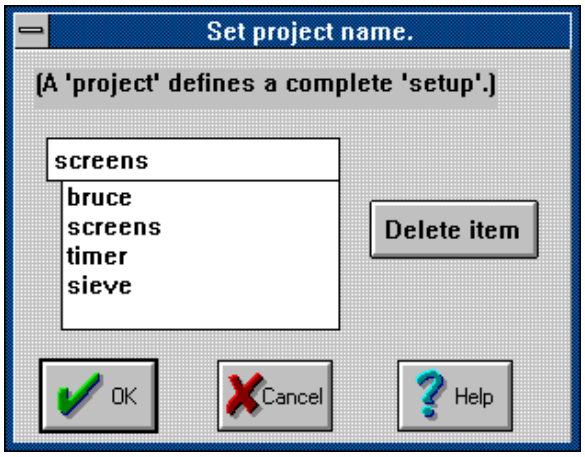


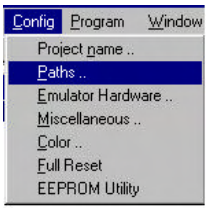
Figure 2. Set Project Name Dialog Box

To add a project, type the new name over the current name. Because the project name is used as the body of a DOS file name, do not use characters in the name that cannot be used in a file name (like a space character). The new project will inherit all the settings from the old project. Projects are deleted by highlighting the project name you want to delete and then clicking on the Delete item button.

Figure 3. EMUL12™-PC/BDM Title Bar

The name of the current project appears in the emulator software title bar. Figure is an example of the **EMUL12** title bar for the project named **SCREENS** (used to create the screen shots for this manual).

Setting the Paths ..



The next item in the **Config** menu is **Paths** .. which opens the dialog box shown in Figure of the manual. The emulator uses these directories to find the files it needs.

Each of these fields can hold up to 1024 characters. Each directory in the path must be separated by a semi-colon (;) just like MS DOS path names. By default, The **User load modules:** field will contain the directory from the last loaded object file, and the Emulator internal files: field will contain the directory where the emulator files were installed.

The **Load path:** directory is the default directory searched for HEX, IEEE 695 files and absolute object files. Any directory can be specified when loading a module, but the directory shown here is the default. The **.ext** field specifies the default file extension. Files in the default directory with this extension will be shown in the **Load code..** dialog box.

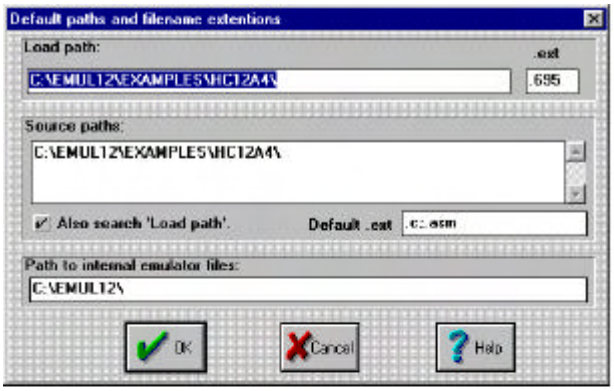


Figure 4. Paths Dialog Box

With many compilers, the full path name of the source file is contained within the object file. Linked object files consisting of several linked objects will, correspondingly, have several source file names and paths. If that source file name exists in the object file that EMUL12™-PC/BDM is loading, the debugger will look for that source file when updating the **Source** window.

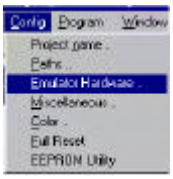
The second field, **Source paths:** identifies other directories to search for missing source files not identified in the object file, or files moved since the compile. The directories in this

field must be entered by the user. Once entered, directories will stay here until removed by the user. The small check box, when checked, will tell EMUL12™-PC/BDM to look for source files in the **Load path:** directory as well. Simple projects may have all the source and object files in the same directory (the **Load path:**) and may not need any directories in the **Source paths:** field.

*Note: The default ".ext" field specifies the source file extension. If your C modules have the extension ".c", enter that. To see assembler source (.asm) in the **Source** window, enter ".asm" .*

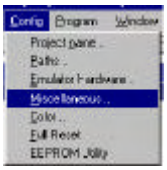
The **Emulator internal files:** field will be set during the installation and probably will not need to be changed. **Emulator internal files:** is the directory the application uses to find the various support files that are part of the EMUL12™-PC/BDM software such as register definition files and dynamically loaded libraries. Normally, the installation program will set this to the proper directory. If you copy or move EMUL12™-PC/BDM to a new directory or disk drive, remember to change this field also.

Emulator Hardware Configuration



The **Emulator Hardware ..** menu item configures the software to correctly communicate with the hardware and your target processor. It presents the Hardware Setup window. Refer to page 4 for a detailed description of how to initialize the Hardware Setup window.

Miscellaneous Configuration



The **Miscellaneous** item in the **Config** menu opens a dialog box that controls special features of EMUL12™-PC:

(1) when and if automatic resets occur.

(2) optional reset vector values.

(3) the source code address range for limiting where breakpoints are set.

By default, the emulator resets the controller when the EMUL12™-PC/BDM software is started. The **Reset chip at start up:** radio button can disable this reset which may be helpful during particularly difficult or unusual debugging circumstances.

Figure 5. Miscellaneous Setup Dialog Box

Even though the next field appears to work, it is not implemented yet. Writes to memory are always read back. For information about the status of this feature, contact customer support.

Figure 6. One Feature Not Yet Implemented

The **Tab Size** field in the **Miscellaneous** menu dialog box allows each tab in the source files to be translated to various number of spaces as determined by this field. The default for this field is 3 (each tab in the source file is displayed as 3 spaces in the **Source** window).

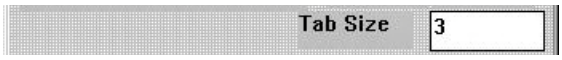


Figure 7. Tab Size Field

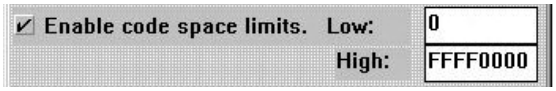


Figure 8. Code Space Limits

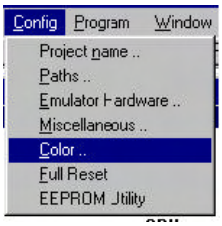
In some symbols files, symbols are not identified as Program space variables or as Data space variables. If this is true of your file, you may see the EMUL12™-PC/BDM software try to set breakpoints in your variable address range. To prevent this, check the **Enable code space limits** box and set the Low and High addresses to encompass just the instructions. Configured this way, EMUL12™-PC/BDM will not put breakpoints outside that address range.

Figure 9. Forcing Reset Vectors

When the **Override at Reset** boxes are checked and the fields contain addresses (in hexadecimal notation), those values will be written to the controller's program counter and stack pointer every time EMUL12™-PC/BDM resets the controller. If you have some test code at an address that you want to execute, filling in this field will force the program counter to the specified value each time you reset the controller. Similarly, to run the test code right after a reset, the stack pointer register must have a legitimate value. This field will conveniently force the stack pointer to a specified value after reset without having to run your start-up code.

Warning:	***Using the Stack Pointer field will set the stack pointer for you which will not happen on a stand-alone target. If you use this field, your start-up code must also set the stack pointer every time the controller is reset or your application will behave differently. In a similar way, using the Program Counter field will set the Program counter to your specified address. To allow the same thing to happen in a stand-alone target, you must program the same address to the Reset Vector in your target Vector table (address ffffeh).***

Window Colors



Under the **Config** menu is the **Color ..** menu item. Open this dialog box to set the colors of different kinds of EMUL12™-PC/BDM child windows. For example, all **Program** windows can be set to have a dark blue background with white text to differentiate them from other kinds of windows. At the same time, all **Data** windows can be green with Black text, and all **Source** windows set to have white background and red text. It is possible to make the screen quite attractive.

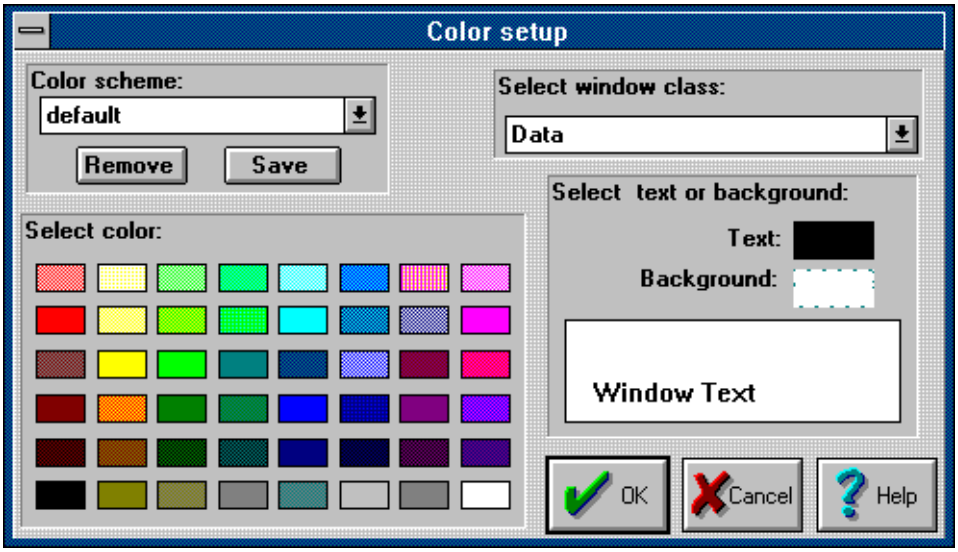


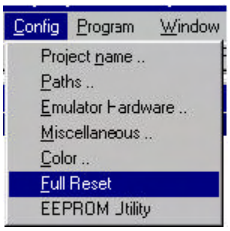
Figure 10. The Color Setup Dialog Box

For each window class that you wish to change, select the window class from the **Select window class** drop list. While that class name is showing in that field, the colors you select will be assigned to that class of windows.

After you have set all the colors the way you want them, you can name a color scheme by typing the name in the **Color scheme** field and then click on the **Save** button. This color scheme can then be recalled by selecting it from the drop list of color schemes.

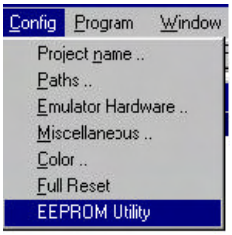
Note: Not all combinations of background and foreground colors are possible. EMUL12™-PC/BDM is constrained by the same limits as MS Windows itself, and is affected by the color palette chosen in the Windows Control Panel program. No matter what colors you select from this palette, the example text pane will show you the colors that will actually be used by EMUL12™-PC/BDM.

Reset vs. Full Reset



Under all circumstances that you will encounter, a Full Reset is the same as clicking on the **Reset** button in the speed bar, selecting **Chip Reset** from the **Run** menu, or pressing **<Ctrl>F2**. With both kinds of reset, the controller is reset by pulling the reset line low. The controller is immediately halted.

EEPROM Utility



The EEPROM Utility menu item opens a dialog box as shown in Figure allowing the user to configure the software to blank-check, erase, program and verify the internal EEPROM memory of the target HC12 (if the target processor you are using includes an internal EEPROM).

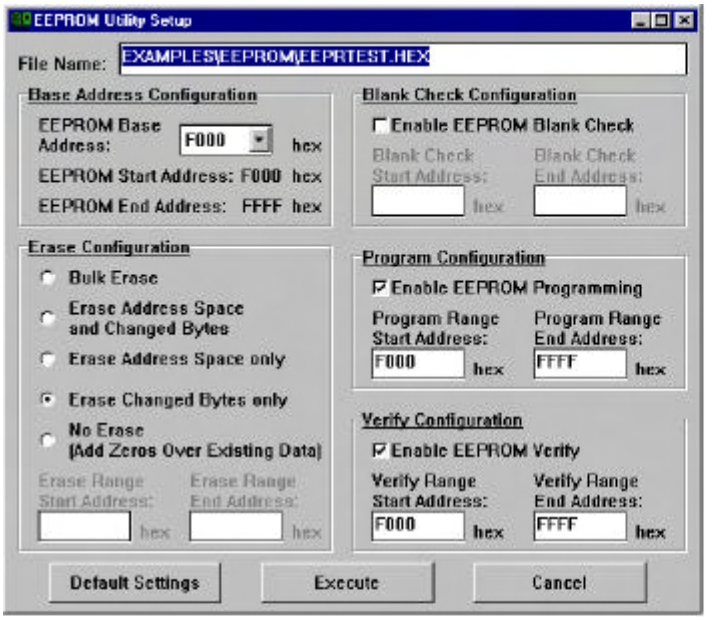


Figure 11. EEPROM Utility Setup Window

The EEPROM Utility uses all of the loaders that the EMUL12-PC/BDM software uses, allowing the utility to load data to be programmed to the internal EEPROM and to verify and compare it to the content of the internal EEPROM. To specify a file to be used, type the file name of the file containing the code (including the path), in the **File Name** field.

The **EEPROM Base Address** allows the user to choose the base address of the internal EEPROM which will be used during the EEPROM Utility operation. The base address is configured using the HC12 - INITEE register. The **EEPROM Start Address** and **EEPROM End Address** fields display the EEPROM range for the current **EEPROM Base Address** value.

When you change the **EEPROM Base Address,** the EEPROM range fields and all of the other enabled range fields **(Erase Range, Blank-Check Range, Program Range** and **Verify Range)** are automatically updated according to the **EEPROM Start Address** and **EEPROM End Address.**

The order in which the EEPROM Utility executes is:

1. The EEPROM Utility checks the validity of the setting entered in the **EEPROM Utility Setup** window.
2. The EEPROM is configured to enable program and erase, and to be located according to the **EEPROM Base Address** field.
3. If a **Bulk Erase** or an **Address Space Erase** is requested, then it is executed at this point. In the case of an address range erase, the address range that begins in the **'Erase Range Start Address'** and ends in the **'Erase Range End Address'** will be erased. In the event of a **Bulk Erase,** all EEPROM bytes will be erased.
4. If the **'Enable EEPROM Blank Check'** check box is enabled, the EEPROM address range between **'Blank Check Start Address'** and **'Blank Check End Address'** will be blank checked (every byte in the range will be compared against ffh).
5. If the **'Enable EEPROM Program'** check box is enabled, the EEPROM address range between **'Program Range Start Address'** and **'Program Range End Address'** will be programmed (using the loaded data from the file specified in the **Load File** field). If there is a need to erase a byte in order to program it with the new data, an erase will occur if **'Erase Changed Bytes only'** or **'Erase Address Space and Changed Bytes'** is selected.
6. If the **'Enable EEPROM Verify'** check box is enabled, the EEPROM address range between **'Verify Range Start Address'** and **'Verify Range End Address'** will be verified (according to the loaded data from the file specified in the **‘File Name’** field).

The **Erase Configuration** group offers five different option of Erase:

1. **Bulk Erase:** All bytes of the target EEPROM memory will be erased.
2. **Erase Address Space Only:** The address range specified by the **'Erase Range Start Address'** and **'Erase Range End Address'** will be erased. This option will not enable erase of bytes during programming as in the **No Erase** option (see details in paragraph 5 below).

3. **Erase Changed Bytes only:** When programming the EEPROM, if a byte of the EEPROM already contains a value other than ffh and it should be erased before it can be programmed with the new data, enable erasing this byte first, and then program it with the new data.

4. **Erase Address Range and Changed Bytes:** A combination of enabling both **Address Range Erase,** and **Changed Bytes Erase** (see details in paragraph 2 & 3 above).

5. **No Erase (Add Zeros Over Existing Data):** No erase will be used. If during the program process, there is a byte that need to be erased before it is programmed with the new data, this byte will not be erased. The program will assign the byte with the new value over the existing value of the EEPROM, resulting in adding zeros to the data.

Note: For 'Erase Address Space Only' and 'No Erase' options, the data programmed in the EEPROM may be different from the data specified in the file due to inability to erase bytes that need to be erased first.

The **Program Configuration** group enables programming the address range between the program range start address & end address with the data loaded from the file specified in the **File Name** field. The method used to program the EEPROM is designed so a minimal number of erase and program cycles will be used: If an EEPROM byte has to be programmed with data, it will be programmed again only if its current content is different from the new data to be programmed. Furthermore, the system checks if it is possible to program the new byte of data in the EEPROM without erasing the old data in the byte. This will happen if -- in order to change the byte value from the old value to the new value -- only bits that were logic 1 need to be changed to logic 0 and there are no bits which need to be changed from logic 0 to logic 1. In this case, the new data is programmed over the old data without erasing the byte value first, producing the requested result of the new value being programmed correctly.

If a byte value needs to be changed and an erase is required before the new content is programmed, the system checks whether or not an erase of changed bytes is allowed. Erase of changed bytes is allowed only when using the **'Erase Address Space and Change Bytes'** erase option or the **'Erase Changed Bytes only'** erase option. If the erase of changed bytes is not allowed, then no erase will occur during the programming process, and the new data will be programmed over the existing data resulting in additional zeros programmed over the old data.

Pressing the **Default Settings** button sets the default values in all of the fields to match the target processor you are using, and the operating mode selected in the **Hardware Setup** window.

Pressing the **Execute** button will activate the execution of the EEPROM Utility: The settings of the EEPROM Utility are saved in the ini file. The validity of the values in the different fields is checked (taking into consideration the EEPROM base address and the nature of the EEPROM in the target processor you are using).

If a validity error is found, an error message titled "EEPROM Utility Error' will appear, explaining the nature of the failure. If the validity check passes, then the EEPROM Utility starts executing the blank-check, erase, program and verify as explained above.

If the utility encounters an error or failure during its execution, an error message will appear explaining the nature of the error encountered. If the utility concludes its execution successfully, a conclusion message will appear summarizing what was executed by the utility.

After the EEPROM Utility execution is done, reset is performed to the target processor and the regular execution of the EMUL12 program continues.

The **Cancel** button cancels the last changes made in the **EEPROM Utility Setup** window, and exits to the regular EMUL12 software without making any change to the EEPROM and without saving the settings to the ini file.

The EMUL12 software is supplied with a sample S-REC file named EEPRTEST.HEX, which may be used to test the EEPROM Utility operation and the EEPROM memory of your target. The source files for this S-REC file are also supplied in the 'EXAMPLES\EEPROM\' sub-directory. This sample file was compiled using the Motorola - MCUasm software package.

Menus

The primary means of controlling the debugger, thus the emulation, is through menus. The EMUL12™-PC/BDM menus conform completely with the *Microsoft MDI* standard. Only those menu items that have meaning or can be used with the current selection will highlight when the mouse is pointing to them. Menus are organized to hide items that are out of context.

Most menu items have "Hot Key" equivalents. That is, there is some combination of function keys, character keys, and modifier keys (Control, Shift, or Alt keys) to select most menu items. The Hot Key for each menu item is shown in that menu to the right of the item name, and are also shown below. Where you see "<Alt>FS" as the keyboard shortcut, you should type <Alt>F (hold the Alt key down while you then press the F key) to open the File menu, then press the S key (without the Alt key) to activate the portion of EMUL12™-PC/BDM that writes "S" record files. Holding down the Shift key or turning on CapsLock is not necessary. Even though the keyboard shortcuts are all shown in capital letters, the shortcuts are not case sensitive.

File Menu

Menu Item	Hot Key	Function
Load code	F3	Load a Hex file, an IEEE 695 record file or one of the other supported object files.
Load default CPU symbols	<Alt>FL	Load the default symbols to enhance the display in **Program** windows by converting the addresses of registers into their respective names. Note that loading default symbols may take as long as 40 seconds on some machines.
Save code as ..	<Alt>FS	Write the contents of RAM or ROM to a HEX record file.
Remove Symbols	<Alt>FR	Delete all line number and symbolic information, and close source files.
Show load info ..		Display a window describing the object file last loaded including number of variables, address range loaded, etc.
Preferences	<Alt>FP	Controls the way the emulator loads object files. Also controls how the data is interpreted for formats larger than 8 bits (i.e., LSB or MSB first).
Exit	<Alt>X	Quit the EMUL12™-PC/BDM application.

View/Edit Menu

Menu Item	Hot Key	Function
Copy to clipboard	<Ctrl><Ins>	Copy the text (without formatting or font information) of the entire active window to the clipboard.
User defined symbols	<Alt>VS	This item opens a dialog box that lets you select the module from which you can view symbols.
Default CPU symbols	<Alt>VC	View and edit memory-mapped registers by name and by the bit.
C call stack ..	<Alt>VC	Opens a child window that displays the C call stack and passed parameters needed to reach the current Program Counter.
Evaluate ..	<Ctrl>E	Open a dialog box that evaluates C expressions. Expressions may contain variables. Assignment expressions may change the values of variables.

*Hint: To change the value of a variable, use the **Evaluate** window to evaluate a C assignment expression such as "i=75".*

Menu Item	Hot Key	Function
Inspect ..	<Ctrl>I	Open a dialog box that displays the contents of a single variable, structure, or array in detail.
Add a watch point ..	<Ctrl>W	Open a child window that displays groups of variables that is updated every time emulation halts.
Search..	<Ctrl>S	This menu item opens a dialog box that lets you search the active window for the kind of data displayed in that window. If the **Source** window is active, you can search for text strings within that file. In all other windows that support searching, the search is for a hex pattern.
Search next	<Ctrl>X	The last search defined will be performed again, from the cursor forward.
Search previous	<Ctrl>P	The last search defined will be performed again from the cursor backwards.

Run Menu

Menu Item	Hot Key	Function
Step into	F7	Execute one instruction, including a jump instruction. If a **Source** window is selected, execute all the instructions for one line of source.
Step over	F8	Execute one instruction or all the instructions in a subroutine. If a **Source** window is selected, execute all the instructions for one line of source. Due to some kinds of optimizations, this feature may not always be available.
Animate ..	<Ctrl>F7	Execute instructions continuously and slowly, highlighting each instruction or each line as it is executed.
Go	F9	Begin executing instructions from the current PC at full speed until the next breakpoint.
Go to cursor	F4	Execute the instructions from the PC to the current cursor position.
Go to ..	<Ctrl>F9	Execute the instructions from the PC to the specified address.
Go to return address	<Alt>F9	Execute the instructions from the PC to the next found function return. Due to certain optimizations, this feature may not always be available.
Go FOREVER	<Alt>RF	Execute instructions from the current PC after disabling all breakpoints.
Break Emulation	F9	Suspend execution as if a breakpoint was encountered.
Reset Chip	<Ctrl>F2	Reset CPU without executing any instructions.

Note: EMUL12-PC/BDM implements only software breakpoints (see the following paragraph for details).

Breakpoints Menu

Menu Item	Hot Key	Function
Toggle	F2	Disable or enable existing breakpoints.

At ..	<Alt>F2	Set a breakpoint by address, line, or line in module.
Setup ..	<Alt>BS	Open a breakpoint editing dialog box.
Disable all	<Alt>Bi	Disable all breakpoints from being active while remaining in the list.
Delete All	<Alt>BD	Clear all existing breakpoints.
Break now!	<Ctrl>C	Immediately halt the emulation.

*Note: EMUL12-PC/BDM implements software breakpoints only. Therefore, trying to set a breakpoint or single step in a ROM memory will fail. Hardware breakpoints may be implemented separately if your target has built-in support for hardware breakpoints. Single stepping in the **Program** window may be used regardless of the type of memory you are running from (either ROM or RAM).*

Config Menu

Menu Item	Hot Key	Function
Project name ..		Choose a configuration or project from a list of existing projects, or create a new one.
Paths ..	<Alt>CP	Sets the default directories for finding load files, source files, and emulator files.
Emulator Hardware ..	<Alt>CE	Sets the emulator board address, controller type, mode of operation, and more.
Miscellaneous ..	<Alt>CM	Sets automatic PC & SP reset value, and memory scroll range values.
Color ..	<Alt>CC	Assign colors to windows.
Full Reset	<Alt>CF	Reloads on-pod logic & performs reset.
EEPROM Utility		Configure the EEPROM Utility to blank check, erase, program and verify the on-chip EEPROM

These next eight submenus share one location in the menu bar. The menu displayed corresponds to the kind of child window selected. Selecting a different kind of child window will change which menu is displayed. To select a different window, either use the **Window** menu, or just click the mouse on any part of the desired window.

Program Menu

Menu Item	Hot Key	Function
Address..	<Ctrl>A	Scroll the selected **Program** window to the specified address.
Origin (at program counter)	<Ctrl>O	Scroll the **Program** window to display the PC address.
Set new PC value at cursor	<Ctrl>N	Set the Program Counter to the address at the cursor.
Module	<Ctrl>F3	Open a dialog box that allows quickly scrolling the **Program** window to the start of any module.
Function	<Ctrl>F	Open a window listing all the functions in all modules loaded. Selecting one will scroll the **Program** window to the start of that function.
View source window	<Ctrl>V	Scroll (or open) a **Source** window to show the source at the current Program window cursor.
Toggle breakpoint	F2	Enable or disable a software breakpoint at the cursor.

Source Menu

Menu Item	Hot Key	Function
Address..	<Ctrl>A	Scroll the selected **Source** window to the specified address, which may be a function name or a label.
Origin (at program counter)	<Ctrl>O	Scroll the **Source** window to display the Program Counter address.
Set new PC value at cursor	<Ctrl>N	Set the Program Counter to the address at the cursor.
Module	<Ctrl>F3	Open a dialog box that allows quickly scrolling the **Source** window to the start of any module.
Function	<Ctrl>F	Open a window listing all the functions in all modules loaded. Selecting one will scroll the **Source** window to the start of that function.
Call stack ..	<Alt>SC	Open a window that displays the C call stack and passed parameters to reach the current Program

		Counter.
View assembly code	<Ctrl>V	Scroll (or open) a **Program** window to the current program counter (not source window cursor).
Toggle breakpoint	F2	Enable or disable a software breakpoint at the cursor.

Data Menu

Menu Item	Hot Key	Function
Address..	<Ctrl>A	Scroll the selected **Data** window to the specified address.
Edit ..	<Enter>	Alter the contents of the highlighted location.
Block move..	<Ctrl>B	Move a segment of RAM to another location (in RAM).
Fill..	<Ctrl>F	Fill RAM with the specified value or pattern.
Display as..	<Ctrl>D	Set the data display mode (ASCII, hexadecimal bytes, long integers, etc. See page 28 of the manual for the complete list of formats).
Address space ..	<Altl>DA	Set the address space for the selected **Data** window. You may select either data or shadow spaces

Shadow Menu

When the **Shadow** Window is selected, the **Data** menu appears.

Register Menu

Menu Item	Hot Key	Function
Edit		Select a register, then edit it's value by selecting this menu item (or more simply, select a register and then type a new value).
Modify Display		Enables you to choose which registers will be displayed in the **Register** window.

Special Register Menu

Menu Item	Hot Key	Function
Add	<Ins>	Add a new register or register bit to be displayed in the window.
Remove	<Del>	Delete the highlighted register from the window.
Edit	<Enter>	Edit the content of the highlighted register.
View/Edit bits..		View and edit the different bits of the highlighted register.

Stack Menu

Menu Item	Hot Key	Function
Parameters in Hex	<ALT>SP	Display the function parameters in hex instead of in their declared type.
Show function	<ALT>SS	Not implemented at this time.

Watch Menu

Menu Item	Hot Key	Function
Add ..	<Insert>	Open a dialog box for adding a variable to the **Watch** window.
Edit ..	<Enter>	Open a dialog box for editing an existing variable in the **Watch** window.
Remove ..	<Delete>	Delete the selected variable from the **Watch** window.

Window Menu

The **Window** menu items open new windows, close existing windows, select windows, and arrange windows on the screen.

Menu Item	Hot Key	Function
Open a new program window	<Alt>WNP	Open a new **Program** window.
Open a new data	<Alt>WND	Open a **Data** window.

window		
Open a new source code window	<Alt>WNS	Open a **Source** window.
Open a new register window	<Alt>WNR	If one is open, it will ask "Are you sure?"
Open a Special Registers window	<Alt>WNE	Open a Special Registers window.
Open a Watch window	<Alt>WNW	Open a Watch window.
Toggle help line	<Alt>WH	Turn on or off the text at the bottom of the EMUL12™-PC/BDM window.
Repaint	<Ctrl>R	Repaints the screen.
Tile windows	<Alt>WT	Resize and arrange the windows within the EMUL12™-PC/BDM application.
Cascade windows	<Alt>WC	Resize and overlap the windows within the EMUL12™-PC/BDM application.
Arrange Icons	<Alt>WA	Line up any minimized EMUL12™-PC/BDM icons at the bottom of the main window.
Zoom	F5	Expand the selected window to fill the EMUL12™-PC/BDM window.
Next window	<Ctrl>F6	Change the currently selected (highlighted) window.
Close	<Alt>F4	Close the currently selected window.

Below the **Close** menu item, there is one menu item for each open window, and the active window will be checked. Selecting one of these items will open the window if it is minimized to icon size, and activate it.

Help Menu

Selecting the **Info ..** menu item will open a box that displays the application version number and date. Please have this information handy when calling for support.

Dialog Boxes

Many menu selections open dialog boxes that allow you to input more specific information. Some of these dialog boxes are described above next to their menu items. The rest are described in this section.

Child Windows

There are eight primary child windows created by EMUL12™-PC: **Program** window, **Data** or **Shadow** window, **Inspect** window, **Source** window, a **Registers** window, a **Special Registers** window, a **Call Stack** window, and **Watch** window. All of these windows are opened by selecting the corresponding item in the **Window** menu.

Any number of child windows may be open at the same time. Any number of child windows can overlap but only one child window is active (has the focus) at a time. Some may be scrolled and resized to view any address desired. Their locations and sizes are saved to the current project file when EMUL12™-PC/BDM exits, and will be restored when the software restarts.

Each child window has a corresponding menu that appears between the **Config** menu and the **Window** menu. The menu contains items that only make sense within the context of that window. This window-specific menu will also appear at the cursor when you click with the Right mouse button in the body of the active window.

Register Windows

The **Registers** window displays the CPU registers. All registers are displayed in hexadecimal notation. Clicking anywhere in the **Registers** window will select that window (make it the active window) and right-clicking brings up the **Registers** menu. The operations supported in the **Registers** window are editing register contents, and selecting which of the registers will be displayed.

Special Registers Window

The **Special Registers** window enables you to view or edit the HC12 Special Function Registers (memory Mapped registers).

Pressing the ‘Ins’ button will enable you to add a new register or register bit to be displayed. Pressing the ‘del’ button will delete a register from the **Special Registers** window. Pressing ‘Enter’ will enable you to change the content of the highlighted register.

Data and Shadow RAM Windows

Use **Data** windows to examine or modify target memory directly. The **Data** window cannot be updated while the emulation is running. Instead, asterisks will be displayed until the next time the controller starts executing monitor code.

Data can be displayed or modified in various formats as shown in Figure 12.

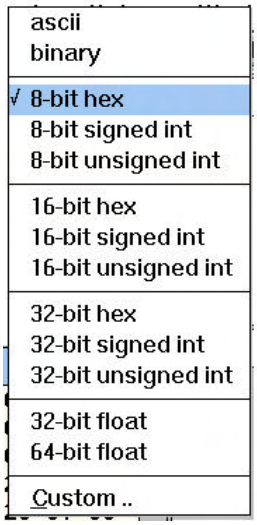


Figure 12. Data Display Formats

*Note: 32 and 64 bit IEEE_754 floating point numbers must be word aligned. Some compilers support packed structures that can have floating point fields that start on an odd address. These fields will not be displayed properly in a **Data** window.*

Selecting any **Data** window displays the **Data** menu which supports filling memory, jumping the selected window to a specific address, setting an address space, and setting the display mode (hex, ASCII, etc.) options .

Changing a value at any memory location is as easy as selecting the byte, word, or long word to change and then typing the new value. The first character you type will open a small data entry window, shown in Figure 13[1].

Figure 13. Editing Memory with a Data Window

Always enter the new data in the same format that the data is displayed. If the **Data** window is displaying ASCII characters, type the new character (not a string) in ASCII. If the **Data** window is displaying signed integers, enter the new value as a decimal number. Symbols are supported but their type is ignored.

If you display the data in bytes, only a byte will be written to memory for each update. In other words, updating one byte uses a single bus cycle that is one byte wide.

[1] Editing in a window displaying memory as ASCII characters will not open the Edit data dialog box. The new character will simply replace the one highlighted.

Custom Display Format

Selecting the custom format option opens a dialog box that lets you input a C printf format string. All standard C formats are allowed, including the newline character. If you are trying to display odd address integers or floating point numbers, you must use the custom display format.

Shadow Window

A **Shadow** window is much like a **Data** window. You may turn a **Data** window to a **Shadow** window by choosing an active **Data** window and selecting the **Shadow** menu item under the **Data - Space** menu item. The **Shadow** window displays the actual memory of your target. The only difference between the **Shadow** window and the **Data** window is the **Shadow** window continues to read the data from your target also when the target is running, while the **Data** window does not display the data when the target is running (only when the target program has breaked).

The Shadow window also enables you to write a new data to the target memory both when the target is breaked and on the fly. Writing new data to a memory location is as easy as selecting the memory location in the Shadow window, then typing the new data to be written (see page 28 for details). This feature allows you to write data to the target memory during emulation.

Warning:	***The use of a shadow memory when the target is running might steal cycles from the target in order to read the memory content. The way that the HC12 BDM is designed, when the target is running and a memory read is requested through the BDM, the HC12 starts looking for a dead cycle (during which it does not access the memory). If it finds a dead cycle within 128 ECLK cycles, it uses it to read the required memory content for the BDM, otherwise it holds the cpu of the HC12 for one (or two) bus cycles to read the requested memory location for the BDM. In almost all cases, this bus stealing of the Shadow window is insignificant for the time stealing of the HC12. In the worst case, it would steal time equivalent to 2 bus cycles (to read a word with a bus width of 8 bit) once every 1280 ECLK cycles (and it is very unlikely that this will actually occur). For systems where the actual read of a memory address performs an "additional action" such as clearing a flag, etc., you must consider the fact that the use of shadow memory will have implications on the "additional action".***

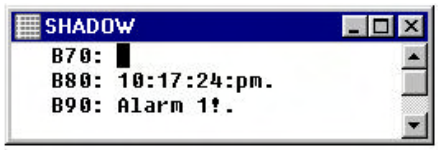


Figure 14. Shadow Example

Shadow windows display values in all the same formats as **Data** windows. This is controlled with the **Data** menu that is presented when a **Shadow** window is selected.

Usually a **Data** window and a **Shadow** window will display the same value for the same address. This is not true when emulation is running (because updating the **Data** window is suspended while running at full speed).

Program Windows

A **Program** window disassembles and displays code memory. One line in the **Program** window is always highlighted. This is the cursor. The color of the highlighting and the window depend upon how you have configured your color settings. (See page 14 for information about how to change the color settings.) Use the cursor to set and disable software breakpoints, set the program counter, and invoke the in-line assembler.

The first column is the hexadecimal address. If the address is highlighted, there is a breakpoint at that address. You may set or inactivate a breakpoint by clicking on the address. The second column is the hexadecimal value at that address. Between the address and the hexadecimal data may be an arrow pointing to the right, indicating the current program counter. The third column contains the disassembled instructions and operands.

A comment will sometimes appear to the right of the highlighted instruction. The comment displayed is a function of the kind of instruction and is a hint about what will happen when the instruction is executed. For example, if the highlighted instruction will change the contents of memory, the hint will contain the value about to be overwritten.

Program windows can control the emulation. To set a software breakpoint, click once on the address portion of the instruction where you want the break. Or, you may click once on the desired instruction (to highlight that instruction) and then click on it again to highlight the address. A breakpoint is indicated by displaying the address with white letters on a black or dark background. This second mouse click (not a double click) creates the breakpoint. To deactivate (not delete) that breakpoint, click again on the same instruction. The address will no longer be highlighted and the breakpoint will be inactive. To delete the breakpoint, use the **Setup ..** dialog box from the **Breakpoints** menu. Any highlighted instruction can be a temporary breakpoint. The Run menu item Go until cursor will use the cursor as a temporary breakpoint.

In-line Assembler

The in-line assembler is easy to use; simply highlight the instruction or address you wish to change in the **Program** window and type. The first character typed will open an edit dialog box to display the characters you type and allow you to edit your assembler source line. Once the source line is as you want it, press <Enter>.

The in-line assembler will translate the input line according to the syntax described in the 68HC12 data book and replace the former opcode(s) and data with the new opcode(s) and data. Note that the assembler will write as many bytes as required for the new instruction. This may overwrite part or all of subsequent instructions. Be sure to examine the subsequent instructions as well as the new instructions for correctness.

IN LINE ASSEMBLER FEATURES

In order to use the in line assembler to your satisfaction, you may need to know the following:

1. Numbers and addresses are interpreted as a decimal number or address as a default.

2. If you would like to enter a hex number or address, you need to type the '$' sign in front of the address or number. Forgetting to put the '$' sign will result in an error message (if the number you have typed in includes one of the letters A-F) or in the decimal equivalent number (if the number you have typed in includes only the digits 0-9). Remember to always use the '$' sign for hex numbers.

3. To use immediate addressing, you must use the '#' sign in front of the immediate data. (If you would like to use hex immediate data, type the '#' sign first, then the '$' sign, and then the requested hex number.)

4. Using the extended addressing mode is selected by default when you specify an address in an instruction. In order to use the direct addressing (to use the short one byte extension, to access an address in the range 0 - ffh) rather than the extended addressing, enter the letter 'd' or 'D' in front of the address you would like to use. For example, typing 'adda d$d5' will produce the two byte opcode "9b & d5" and typing 'adda $d5' will produce the three byte opcode "bb, 00 & d5".

5. When using the constant indexed addressing mode, the in line assembly will implement the shortest constant indexed addressing mode that can be used. If the constant is in the range of -16 to 15, 5-bit constant offset indexed addressing will be used. If the constant is in the range of -256 to 255, 9-bit constant offset indexed addressing will be used. If the constant is out of the range of -256 to 255, 16-bit constant offset indexed addressing will be used. In case you are using a small constant offset but would like to force the in line assembly to use the 9 bit constant offset addressing or the 16 bit constant offset addressing, you should do the following:

   - To force the use in the 9 bit constant offset, the letter 'e' or 'E' should be typed before the constant.
   - To force the use in the 16 bit constant offset, the letter 'l' or 'L' should be typed before the constant.
   - For example: typing 'ldaa -$a,x' will produce the two byte opcode "a6 & 16"; typing 'ldaa e-$a,x' will produce the three byte opcode "a6, e1 & f6", and typing 'ldaa l-$a,x' will produce the four byte opcode "a6, e2, ff & f6".

6. Using the TFR or EXG instruction enables you to use the A, B, CCR, T3, D, X, Y & SP registers as source registers, and the A, B, CCR, T2, D, X, Y & SP registers as destination registers.

7. If you would like to implement one of the page 2 unimplemented opcodes, you may type 'trap' and the number of the second page opcode you would like to use. The result will be a two byte opcode of 18 and the number you have specified. For example, typing 'trap $56' produce the two byte unimplemented opcode "18 & 56".

Source Windows

The **Source** window displays the C source (or assembler source if the assembler supports source line debugging) of the module containing the Program Counter. Like a **Program** window, a **Source** window displays the source text, line numbers, a cursor (the blinking underline), and a small arrow between the line numbers and the source text to indicate the current Program Counter value.

After each single step, and during each animation pause, the **Source** window scrolls to show the source line that generated the instruction pointed to by the new Program Counter, if it was generated by a source line.

Displaying and toggling software breakpoints in **Source** windows is different than in **Program** windows. In **Source** windows, breakpoints are displayed by inverting (or highlighting) the entire source line. In **Program** windows, only the address is highlighted. In **Source** windows, a single click on any line number (or address in the **Program** window) will toggle the breakpoint. In both kinds of windows, pressing **F2** will toggle a breakpoint on the highlighted instruction.

When a **Source** window appears blank with the window title "Source", it usually means that the program counter is pointing to instructions derived from a module with no debugging information. As soon as the PC points to an instruction from a C module or assembly module with line number symbols, the **Source** window will show that text, and the title on the window will change from "Source" to the name of the source file being displayed.

The simplest way to find the first line of source is to reset the controller, click on the **Source** window title bar to select it, and then execute a single step by pressing the **F7** key (or by clicking on the **Step** button on the speed bar).

When the **Program** window is selected, a single step means a single opcode. The same is true for animated execution: a pause occurs after every opcode is executed. When the **Source** window is selected, a single step means a single source line. Animation will execute faster when the **Source** window is selected than when the **Program** window is selected because most source lines compile into more than one machine instruction. If the animation is running faster or slower than you expect, or if single stepping executes more or fewer instructions than you expect, visually confirm that the selected window is the one you want to be selected. If in doubt about which window is selected, click on the title bar of the window you wish to be selected.

Other Windows

Three more child windows used for high level debugging in C are available: the **Inspect** window, the **Watch** window, and the **Stack** window . These windows are opened by selecting their respective items in the **View/Edit** menu. Like the other child windows, selecting one of these open windows will bring a corresponding menu up between the **Config** and **Window** menus.

INSPECT WINDOW

The **Inspect** window displays a single variable, or possibly modifies that variable. To open an Inspect window, either select the Inspect .. menu item in the **View/Edit** menu or double click in the **Source** window on the variable you would like to inspect. Double-click in an open Inspect window on a structure member or array element to open an **Inspect** window detailing that field.

The **Inspect** window can stay open just like a **Data** or **Watch** window, and it will be updated whenever the application stops. The variable being displayed may be part of an equation written following the rules of C that produces a single scalar answer.

---

*Note: If you have an open **Inspect** window with an assignment statement, every time the emulator stops executing, the expression will be evaluated and the variable will be updated. The variable will appear as though your application is not changing it while the emulator is running.*

WATCH WINDOW

The **Watch** window displays multiple variables being watched, one variable per line. Any local variable in the **Watch** window that is not in scope will be displayed with three question marks instead of its value.

STACK WINDOW

The **Stack** window displays the "call stack," or the list of functions called to reach the current point in the application, and the current value of parameters passed to them .

Addresses are displayed and entered using hexadecimal notation or global symbol names. In all windows (excluding Inspect windows,) values may be edited by selecting that value (with the mouse or cursor keys) and then typing.

Note: Symbol names are case sensitive. If a symbol cannot be found, try the same name with a different case. Also note that some assemblers shift all symbols to uppercase.

Tool Bar

Just below the menu bar is the "Tool Bar" containing icons or buttons that, like Hot Keys, execute frequently needed menu options when clicked. The **Help** button opens the *MS Windows* Help application to the page that describes the current context. The **Reset** button resets the controller. The **Step** button emulates one source line or opcode depending upon which window was last active. The **Go** button starts full speed emulation that will continue until a break occurs. While emulating, the **Go** button changes to **Break**, and halts emulation when clicked.

Figure 15. The Tool Bar

Help Line

At the bottom of the EMUL12™-PC/BDM window is a line of text that, depending upon the context, explains what the selected item is or what it does. This kind of context-sensitive help is turned on and off with the **Toggle help line** item in the **Windows** menu.

Chapter 2 : Setting the Hardware

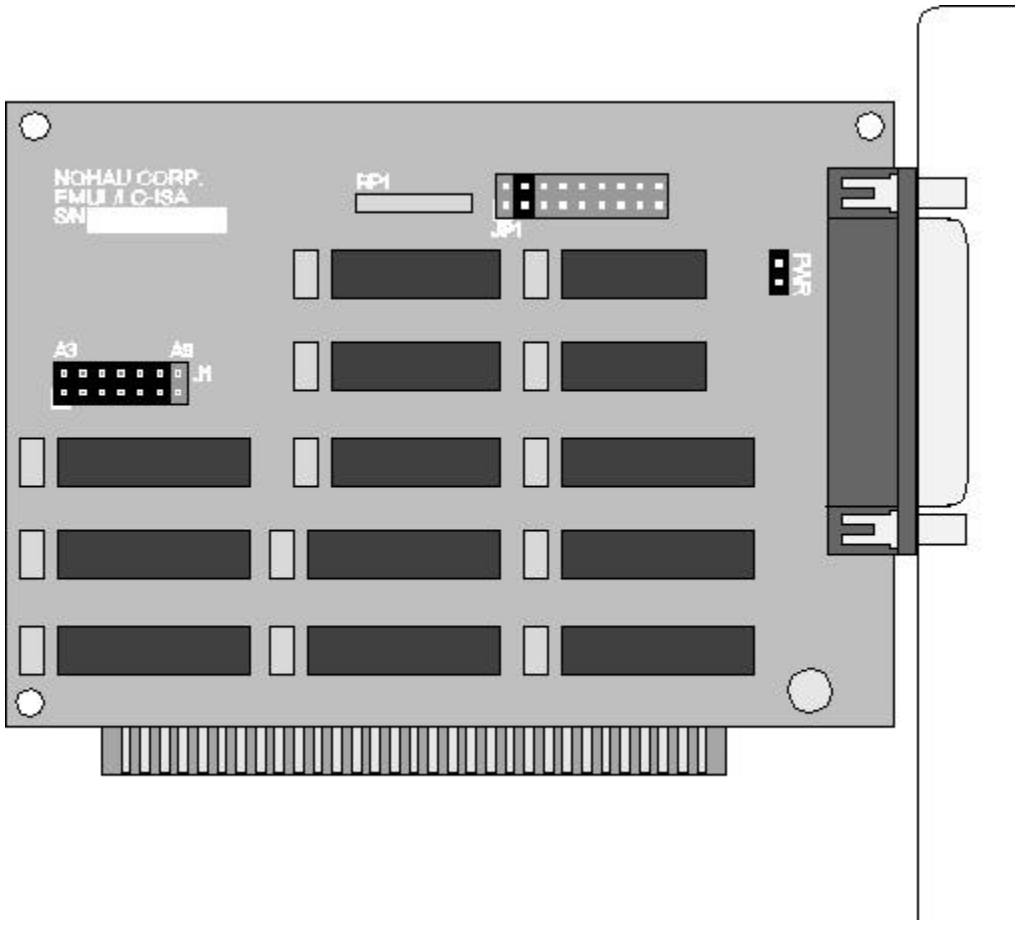


Figure 16. EMUL12-PC/BDM PC Plug-In Card (EMUL-PC/LC-ISA)

The EMUL12-PC/BDM board is an 8 bit P.C. card that fits into any slot (except PCI). The jumpers on the emulator board control three things: (1). the port address used to communicate between the BDM pod and the EMUL12 software, (2). the communication rate between the BDM pod and the PC, and (3). whether or not power is provided to the BDM pod. These are all described in more detail below.

Detailed Installation Instructions

Setting the I/O address jumpers -- J1

The EMUL12-PC/BDM requires 8 consecutive I/O addresses from the P.C. I/O address space (0 Hex -- 3FF Hex) that begin on an address that is a multiple of 8. Set the emulator board address using the jumpers in header J1. These addresses must not conflict with any other I/O device. Each pair of pins in J1 represents one bit in the 10 bit address. Address bits 0, 1, and 2 represent addresses within the 8 consecutive

addresses and do not have pin pairs to represent them. This leaves 7 address bits (pin pairs) to set with jumpers. Shorting two pins represents a 0 in the address. A pair of pins with no jumper represents a 1. Below are 4 examples where the Least Significant Bit is on the left, as it is on the board, if you are holding the board so you can read the silk-screened labels, with the 25 pin D connector on the right.

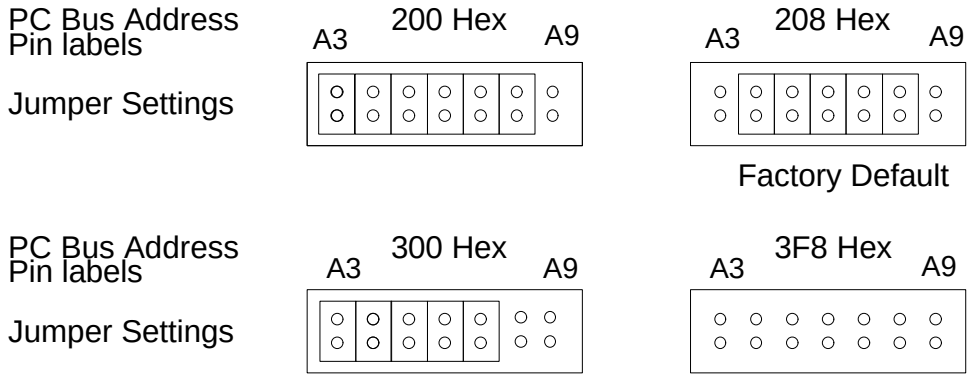


Figure 17. Emulator Header J1

Setting the Communication Rate to the BDM Pod

The EMUL12-PC/BDM uses the PC ISA bus clock to determine the data rate for the communication between the PC plug-in board and the BDM pod. The frequency of the PC ISA bus clock is approximately 8 MHz. The position of the jumper on the header JP1 determines by what factor the PC ISA bus clock will be divided to form the clock for the communication rate between the PC plug-in board and the BDM pod.

Note: Only one jumper should be used on this header.

The header JP1 and the communication rate chosen for each of the jumper positions appears in the following figure.

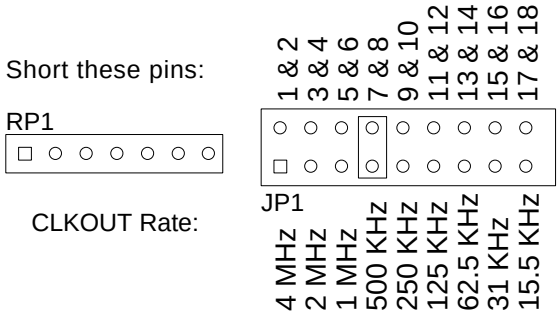


Figure 18. Header JP1

Note: The pins on JP1 are not numbered on the board. The figure above shows the orientation of both JP1 and RP1 as they appear on the emulator board. Both pin 1 holes are shown as square, as they are on the emulator board.

The jumper on the JP1 connector is factory configured to the 4th position of JP1 as appears in **Error! Reference source not found.**, which selects the maximum allowed communication rate between the PC plug-in board and the BDM pod (500KHz).

Warning: Trying to configure the communication rate to be higher than 500KHz (by moving the JP1 jumper to the 4MHz, 2MHz or 1 MHz position) will cause a communication error between the PC plug in card and the BDM pod. If you would like to use a data rate lower than 500KHz, you may do so by choosing one of the higher positions of JP1. This will result in a lower communication rate which will influence the update time of the data in your EMUL12 application.

The communication rate between the PC plug-in card and the BDM pod is completely independent of the communication rate between your target and the BDM pod (through the BDM channel). Therefore, there is no reason to change the configuration of the JP1 jumper from its original configuration.

The PWR Header -- JP2

The third header on the emulator board is the PWR header, which is also labeled JP2. With this jumper in place, +5 volts is supplied to the BDM pod from the P.C. power supply. This jumper should always be in place to power the BDM pod.

Note: EMUL12-PC/BDM has been designed to be compatible with 2.5 to 5 volt target designs. Therefore, the target power should be connected to the BDM pod through the 6 pin BERG connector, in addition to this power from the PC. The power from the target is used to power up the output drivers of the BDM pod, so the voltage seen by the target will be compatible with the target power supply. There is no danger of shorting the target power supply to the PC power supply.

Figure 19. The BDM POD

Background Debug Mode connector

The 6 pin connector is the BERG “Background Debug Mode” connector. This connector standard (defined by Motorola) allows the EMUL12-PC/BDM to debug and control target boards (with a standard BDM header) with no physical room for a pod board and at a lower cost than a full-featured emulator.

This BDM connector is designed to be plugged into single wire BDM targets such as the HC12. (It is not designed to emulate or debug the earlier non-single wire BDM targets, such as CPU16 -- 68HC16 or CPU32 -- 683xx families).

Figure 5 shows the standard pin-out of the 6 pin BERG connector on the target side, looking down onto the board with the pins pointing up towards you.

BKGD	1	2	GND
N. C.	3	4	RESET
N. C.	5	6	VDD

Figure 20. 6-Pin BDM Connector - Target Side View

Note: The VDD pin is used to take your target voltage and use it to power the output drivers of the BDM pod. Therefore, the voltage you will get on the outputs of the BDM connector will be compatible with your target power supply voltage.

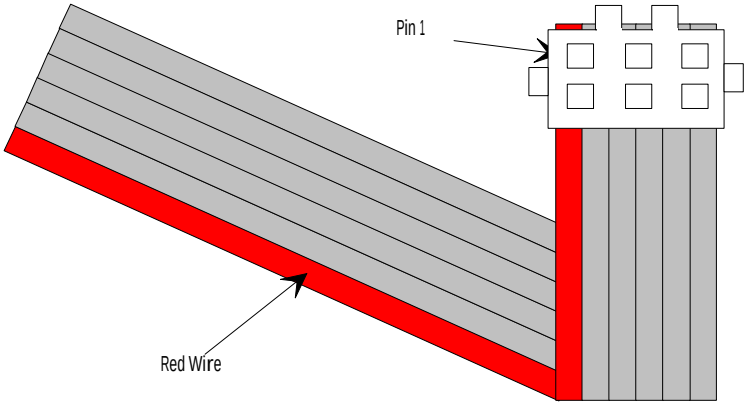


Figure 21. BDM Connector Detail, End View

MODA & MODB

The 2 extra yellow and red wires coming out of the BDM pod are to be connected to the MODA (yellow wire) and MODB (red wire) of the target HC12 chip in order to force the target chip to the required operating mode. (EMUL12-PC/BDM forces your target to Special Single Chip mode, and then switches to the required operating mode).

If you would like to use the target PE5/MODA & PE6/MODB pins as I/O port pins, you may do so under the following restrictions:

- During an external RESET to the target HC12 and 500nSEC after RESET is de-asserted, you may not drive the MODA and MODB pins; during this period, the BDM POD is driving these pins.

Therefore, if you would like to use these pins as input pins, you must make sure that the external driving logic is tri-stated during the discussed period. If you would like to use these pins as output pins (if you are using the RESET and GO option that will be available in the future), you must make sure these pins are not turned to output I/O pins within the first 500nSEC after reset.

To be able to use your target in stand-alone mode when you finish debugging your software using EMUL12-PC/BDM (without having to change your target hardware and without conflicting with the signals driven by the BDM Pod during the use of EMUL12-PC/BDM), you can place pull-up and pull-down resistors (4.7Kohm or higher) on the BKGD, MODA and MODB pins rather than tie these pins directly to VCC or GND. This will prevent a conflict between the BDM POD drivers and your target connections for these pins.

After the jumpers are set as described above, with the P.C. power off, remove the P.C. cover, insert the PC plug-in board into any free P.C. slot (not into PCI slot), close the P.C. cover, connect the cable to the emulator board, connect the BERG connector to the target, connect the two MODA and MODB wires to your target, turn on the P.C., start *Windows*, apply power to the target, and finally start the EMUL12 software.

Notes about Motorola Evaluation Boards

When using one of the Motorola Evaluation boards with the EMUL12-PC/BDM, make sure that the jumpers on the BKGD, MODA and MODB lines are removed, then connect the 6 pin BERG connector and the two MODA, MODB wires to your Evaluation Board.

For more information about emulating either of these evaluation boards or about emulating any other evaluation board, please call or email technical support.
support@icetech.com

Chapter 3: BDM Limitations

For most users and applications, the following limitations will not apply. However, since the EMUL12-PC/BDM is a BDM based emulator, it must have a few limitations which are detailed in this chapter.

Hardware Limitation

1. There are some limitations on the use of the target pins BGND, MODA, MODB and RESET. Preferably, these target pins should be connected directly to the BDM POD, with no logic on your target driving these signals which would conflict with the signals driven to these pins through the BDM POD. The RESET, MODA & MODB signals are driven by the BDM POD only when the EMUL12-PC/BDM system is trying to reset your target processor and 500nSEC after reset is over. At all other times, these BDM POD output signals are tri-stated.

   Therefore, if you would like to use these 3 signals, you may do so when reset is not driven by the BDM POD, and 500nSEC after reset is de-asserted (see details on page 39).

   The BGND signal, on the other hand, is always used by the BDM POD, and thus must be connected directly to the target HC12 chip with no external logic conflicting with it.

NOTE: You may place pull-up or pull-down resistors, 4.7 Kohm or higher, on all 4 signals (BGND, MODA, MODB & RESET).

2. To work properly with the EMUL12-PC/BDM system, the input frequency range of the target is limited to the range 32KHz - 32MHz, and the internal ECLK rate is limited to the range 16KHz - 16MHz.

Register Change Limitation

1. The EMUL12-PC/BDM communication to the target HC12 is made through a single pin in the target HC12: the BGND pin. The communication rate of the data on this pin is defined internally by the target HC12, and there is no way for the EMUL12-PC/BDM to discover when the communication rate has changed or what the changed rate is (because of a change in the clock frequency used by the target HC12).

   Therefore it is necessary for the user to input the correct clock configuration in the Hardware Setup window, and then not change the values of the registers and fields which control the HC12 clock rate (such as PLL activation/deactivation etc.). For more details about registers that may not be changed for this reason, refer to page 4.

2. RTICTL register: In all normal modes of operation, the RTICTL Register is initialized to a different value from its default reset value for normal modes. Bit D5 -

RSBCK is written as a '1' instead of a '0' in order to disable the COP (Watchdog) from running when the program has stopped and the BDM becomes active. For proper operation, if you change this register, be sure to write bit D5 - RSBCK as a logic 1. If you write this bit as a 0, you may see some undesired behavior of the EMUL12 software, i.e., 'losing track' of your application. When using the 'RESET and GO' feature, your application program must write this bit as a logic 1 in one of its first instructions.

Note: In normal operating modes, it is also very important that you either disable the COP at the beginning of your application software or make sure you reset the COP before the timeout is reached. Failure to do so will result in repetitive COP resets, as soon as the COP timeout is reached.

3. The HC12 peripherals often include a control bit to enable or disable the module operation continuance when BDM mode is entered. The timer module, for example, includes in register TSCR the TSBCK bit (bit D5) to enable or disable running the timer while the HC12 enters BDM mode. It is strongly recommended that when you want to use a module which includes such a control bit, you write this bit in your application software to disable the module from continuing to run when BDM mode is entered. This will not have any implications when your target software is running, but may save you a lot of trouble during the debug process.
4. PEAR Register: In Normal Single Chip Mode, the bits in this register that are classified as 'write once' in normal modes are actually 'write never'. In order to be able to write to these bits (in Normal Single Chip Mode), you will have to use the 'Reset and Go' option (when it is implemented).
5. INITRG Register: The INITRG register controls the base address of the HC12 Special Function Registers. If you would like to relocate the SFR base address to another address different from its default address, you may do so by using the 'Internal Register Start Address' field in the **Hardware Setup** window. In order to support some advanced features, you are required not to change the INITRG value in your application software. (For more details refer to page 5, paragraph 8.)

MODA & MODB

The 2 extra wires coming out of the BDM pod (Yellow and Red wires) are to be connected to the MODA (Yellow wire) and MODB (Red wire) of the target HC12 chip in order to force the target chip to the required operating mode (EMUL12-PC/BDM forces your target to Special Single Chip mode and then switches to the required operating mode).

If you would like to use the target PE5/MODA & PE6/MODB pins as I/O port pins, you may do so under the following restrictions: During an external RESET to the target HC12, and 500nSEC after RESET is de-asserted, you may not drive the MODA & MODB pins. During this RESET period and 500nSEC after it is de-asserted, the BDM POD is driving these pin.

Therefore, if you would like to use these pins as input pins, you must make sure that the external driving logic is tri-stated during the discussed period, and if you would like to use this pins as output pins (if you are using the RESET and GO option that will be

available in the future), you must make sure these pins are not turned to output I/O pins during the first 500nSEC after reset.

To enable switching your target to stand-alone mode when you finish debugging your software using EMUL12-PC/BDM (without having to change your target hardware and without conflicting with the signals driven by the BDM POD during the use of EMUL12-PC/BDM), you might want to place pull-up and pull-down resistors (4.7Kohm or higher) on the BKGD, MODA & MODB pins rather than tie these pins directly to VCC or GND. This will allow your target to reset to the required operating mode in stand-alone mode, and prevent a conflict between the BDM POD drivers of these pins and your target connections for these pins.

Chapter 4: Troubleshooting

Troubleshooting Overview

If you have trouble with your emulator, call or email customer support. support@icetech.com support person. If you do, the engineer will likely lead you through the following steps to test for the most common mistakes. To save time, you may also test for the most common reasons that the emulator is not working the way you want yourself

The items to check for below are in order. Start at number 1 and continue until either the emulator works or you have reached the end of the list. Each item is a short version of a description from earlier in this manual. Each item has at least one page number where more details can be found.

Note: In order to check your system, the BDM pod must be connected to your target.

Step 1: PC plug-in board I/O Address

Confirm that the I/O address set by the J1 jumpers on the PC plug-in board is identical to the address entered in the ISA Port field of the **Hardware Setup** window. Check that the I/O address range you are trying to use for EMUL12-PC/BDM is not already used by another hardware device in your system PC. Trying to plug two different devices on the same I/O address range will result in a communication failure between the PC and the BDM pod. (For further information, see page 4 and page 36.)

Step 2: Selecting the Communication rate between the PC and the BDM pod

Confirm that the selected communication rate between the PC plug in card and the BDM pod is not higher than 500KHz. This communication rate is selected by the jumper on the JP1 header in the PC plug in card. The jumper should be set to the fourth position from the left, or to the right of the fourth position from the left. (For further information, see page 36.)

Step 3: BDM Pod Power Supply

Confirm that a jumper is connected on the PWR header in the PC plug in board. (For further information, see page 37.)

Step 4: Hardware Setup Window Settings

Confirm that the type of target selected in the **Pod** field in the **Hardware Setup** window is the same as the type of target controller you are using. Make sure that the frequency entered in the **EXTAL frequency** field in the **Hardware Setup** window is identical to the

Crystal or Clock Oscillator frequency found on the EXTAL pin of your target controller. (For further information, see page 4.)

Step 5: Connection to the target

Make sure the BDM pod is connected to your target processor correctly, using the 6 pin BERG connector and the two MODA & MODB wires. Make sure the pinout you are using for the 6 pin BERG connector is correct. Make sure there is no short on one of the signals RESET, MODA, MODB or BKGD that prevents these signals from being driven by the BDM pod properly. (For further information, see pages 4 and 38.)

Step 6: Target Functioning properly

Make sure you have connected your target properly, the power supply is turned on, and the clock source is functioning correctly.

Step 7: Sample User Program

If you telephone technical support team, you will probably be asked to do the following to enter a sample user program:

1. Click in the **Program** Window
2. Hit <Ctrl>-A
3. Type in address 800
4. Hit <Enter>
5. Type: NOP <Enter>
   NOP <Enter>
   JMP $800 <Enter>
6. Click on the **GO** button in the tool bar.
7. Click on **BREAK**.
8. Make a note of software revision.
9. Make a note of compiler revision.
10. Make a note of serial numbers of boards.

INDEX

N

O

P

R

S

T

U

V

W

Z